Service Function Chain Network Planning through Offline, Online and Infeasibility Restoration Techniques

Ramy Mohamed, Marios Avgeris, Aris Leivadeas, Ioannis Lambadaris, John Chinneck, Todd Morris, Petar Djukic

***Abstract*—Network Function Virtualization (NFV) and Service Function Chaining (SFC) have created a new, flexible, and cost-efficient way of network service offering. Thus, Service (SPs) and Infrastructure Providers (InPs) are updating their product portfolio to incorporate these new technologies. However, placing the network components, called Virtualized Network Functions (VNFs), that constitute the network services on top of the physical infrastructure creates several resource allocation challenges. This paper offers a networking planning solution tool for SP and InPs to allocate resources according to the expected network demand appropriately. In particular, first, we propose an offline and optimal solution that dimensions and exactly allocates the physical resources according to the estimated workload. Following, mispredicted offline solutions are corrected by various online placement solutions that serve the network workload according to the time of its arrival. Finally, to alleviate any resource mismatch between the total capacity of the physical network and the total requested resources of the expected SFCs, a novel feasibility placement approach is proposed that provides suggestions to the InPs on where to place additional resources. Extensive experimentation shows that the proposed planning tool can efficiently preallocate SFCs under different configuration scenarios, while the feasibility restoration tool provides accurate, timely, and cost-efficient remedy solutions for the InPs.**

***Index Terms*—Service Function Chaining, Network Function Virtualization, Network Planning, Feasibility Analysis, VNF Placement**

I. Introduction

The emergence of 5G and beyond networking, as well as the proliferation of real-world Edge Computing settings, allowed for realizing and supporting a wide range of new applications and services, including the Internet of Things (IoT), Virtual and Augmented Reality (VR/AR), as well as autonomous vehicles among others. Along with them came the necessity for providing higher data transfer rates and capacities alongside lower latency in the client-server communication [1]. Towards accomplishing this necessity, one of the prominent technologies that were developed with the next-generation networks is Network Function Virtualization (NFV), which enables previously hardware-bound network functions, such as routing, switching, and security, to be virtualized and run on commodity hardware instead of dedicated appliances [2]. This virtualization trend in networking, subsequently gave birth to the Service Function Chaining (SFC) concept, which introduced the chaining of multiple Virtualized Network Functions (VNFs) together, in a specific order, to create a complete service path [3]. When put together, NFV and SFC can provide network operators with great flexibility and control over traffic routing and service delivery.

However, the VNFs comprising the SFC should be appropriately allocated over the physical infrastructure of the provider by finding the necessary computational and communication resources required for their functioning. For the latter, current and next generation network paradigms are expected to leverage end-to-end solutions taking advantage of the resources found in the recently introduced Edge Computing infrastructure and in the traditional Cloud one. Specifically, the industry has self-adapted in various ways in response to the surge in demand for resource-intensive applications at the network edge. Edge computing has emerged as a primary solution for reducing latency by bringing computation resources closer to users. For instance, 5G network slicing aims to introduce dedicated resources for end-to-end applications, which can be decomposed into virtual functions while using the edge infrastructure for more efficient processing. Moreover, the rise of containerization and microservices has enabled dynamic application scaling and resource allocation adjustments based on real-time demand. Thus, a collaborative approach to edge and cloud processing ensures a balance between rapid response times and the power of centralized data centers [4]. Although these innovations have considerably enhanced service delivery, providing efficient resource planning and management remains an ongoing challenge, especially for SFC deployment.

The particular resource allocation problem, which is called VNF placement and SFC planning need a comprehensive system model, where several key components and parameters need to be identified. For instance, the system model should include a detailed inventory that could be represented as a network topology that includes the functional characteristics of the physical computing and communication nodes as well as those of their physical connections. Similarly, the SFCs can be represented as service requests that specify the sequence and topologies of their interconnected VNFs, their required processing times, the CPU and memory requirements, any latency and bandwidth constraints and so on. By having such a detailed model the infrastructure providers and network operators can find optimal VNF placements and robust SFC

R. Mohamed, M. Avgeris, I. Lambadaris, and J. Chinneck are with the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. A. Leivadeas is with the Deparment of Software and Information Technology Engineering, ÉTS, Montreal, Canada. T. Morris and P. Djukic are with Ciena, Ottawa, Canada E-mails: ramy.mohamed@carleton.ca, mariosavgeris@cunet.carleton.ca, aris.leivadeas@etsmtl.ca, ioannis@sce.carleton.ca, chinneck@sce.carleton.ca, {tomorris, pdjukic}@ciena.com

planning schemes.

These optimal placements and planning approaches depend on the service requirements (e.g., 5G and IoT) and from the capabilities of the infrastructure. Normally, an SFC-enabled infrastructure should cater for performance, scalability, and cost-efficiency optimization, together with resiliency and high availability at all times. To provide such guarantees, proactive and reactive network planning is required. In detail, the Service and Infrastructure Providers (SPs and InPs) should be optimally placing VNFs in their infrastructure to minimize resource utilization, reduce the network latency, and increase the overall network efficiency [5]. This kind of proactive planning results in increased Service Level Agreement (SLA) compliance, scalability, i.e., the network can accommodate new services and users without any performance degradation, while overprovisioning is avoided and the overall network costs are reduced. However, planning the placement in advance, is not always accurate; misplanning in offline solutions can be corrected using reactive, online mechanisms that alleviate resource mismatches between the capacity of the physical network and the requested resources of the expected SFCs [6].

In this article, we aim to address both proactive and reactive network planning in a single planning tool for SPs and InPs, which will facilitate the resource allocation in an online and offline fashion respectively for an edge-cloud infrastructure, according to the expected network demand. In detail, the contribution of our work is fourfold:

- An offline, proactive SFC placement mechanism is proposed that optimally allocates the physical resources of the infrastructure according to the expected workload demands. The objective of this allocation is to minimize the number of edge servers utilized for the VNF placements, as well as the communication cost, in terms of number of links used to interconnect them. In order to avoid computationally intractable situations, we propose two heuristic algorithms for the solution of the formulated Integer Linear Programming (ILP) problem.
- The offline mechanism is followed by an online, reactive SFC placement mechanism, which corrects the potential mispredictions in the incoming workload of the first step. Three algorithms are proposed for this second step, which take into consideration the arrival and departure times of the SFCs, as well as the locations of the end-users that make the requests.
- To give SPs and InPs a better insight into the potential blocking causes for the SFC placement and a more fine-grained way of solving them, a novel infeasibility restoration mechanism is additionally implemented to complement the offline SFC planning step; this mechanism alleviates the resource mismatches between the total capacity of the infrastructure and the requested resources of the expected SFC requests by providing suggestions to the network operators on where to add more resources. Three approaches are examined for this component, which aim to minimize the induced capital and operational expenses (CapEx & OpEx).
- Finally, through detailed numerical results, we evaluate and demonstrate the effectiveness and efficiency of the proposed work in terms of successful SFC placement under different configuration scenarios. Additionally, the feasibility restoration tool is benchmarked on its accuracy and cost minimization.

The rest of this paper is organized as follows. Section II highlights the related work. Section III describes the system model while Sections IV and V present the three main mechanisms comprising the proposed network planning tool, while in-depth explaining the operation of the different algorithms devised for each step. Section VI presents the performance evaluation of the proposed framework and Section VII concludes the paper.

II. Related Work

During the last few years, the problem of planning the placement of VNFs/SFCs in an edge/cloud infrastructure has been tackled in various ways. In this section, we first discuss some recent online and/or offline VNF placement solutions at the edge and the cloud, which however do not treat the infeasibility problem that may be produced due to insufficient resources. Following, we present some relevant works that deal with the feasibility restoration, but do not provide a holistic network planning tool. Finally, we briefly compare our work with the related works in the field and summarize its novelty.

A. Offline and Online VNF placement

Research in VNF placement caters to both planned (offline) and immediate (online) network demands, balancing efficiency with dynamic adaptability. In the latter category that focuses on real-time responsiveness, in [7], a two-step process of VNF placement and migration is employed, using dynamic programming heuristics based on current traffic, while migration acts as a corrective measure for traffic deviations. Using flow characteristics found in production data centers and realistic traffic patterns, the suggested mechanism is shown to outperform the state-of-the-art. Similarly, Zhang et al. in [8] tackle VNF placement and migration simultaneously as an online optimization problem to accommodate new requests in resource-constrained situations. The objective in this work is to maximize the total throughput minus a weighted total VNF migration cost. On a slightly different note, the authors in [9] reformulate the problem as a Markov Decision Process and then use a variant of Q-Learning to solve it in a 6G-enabled edge computing environment. In this way, they manage to perform near-optimal, real-time VNF placement.

On the other hand, offline mechanisms prioritize long-term optimization. For example, Mao et al. in [10] suggest a two-part algorithm that optimizes edge SFC placement to reduce the use of edge and cloud resources, as well as service latency. Here, the requests are dealt in batches, which ensures efficient resource utilization and avoids the redundant data traffic. A similar objective is used in [11], where the authors suggest a two-stage mechanism that first places the VNFs offline and then schedules the users requests among them in real-time. The simulations results here show a significant improvement in the request rejection rate. Dieye et al. also propose a mechanism for the proactive placement of VNFs in [12], where VNFs are deployed in an optimal manner before any request is received

from the end-users to access the service. The objective here is to reduce the cost for the infrastructure. On the other hand, Sun et al. in [13] try to formulate an offline, optimal VNF placement solution based on an ILP model formulation. This mechanism requires knowledge of all the future incoming requests, thus is deemed unrealistic, and an online version of it is proposed as well. The latter still includes a VNF pre-allocation phase.

Combining the good from both worlds, hybrid mechanisms offer both proactive and reactive solutions for VNF placement. An indicative work in this domain is [14], where the authors propose an offline SDN/NFV network planning mechanism with a cost model and a realistic topology description, complemented with an online, resource allocation mechanism based on an experimental validation. The two mechanisms work seamlessly together towards providing a holistic VNF placement solution. Similarly You et al. in [15], propose a load balancing policy named Constrained Min-Max Placement (CMMP) that schedules VNFs in a way similar to the max-min allocation, where they try to assign the highest number of VNFs possible to the least loaded server, offline. Then, they extend this planning mechanism with an online heuristic that speeds up the placement compared to other baselines.

Overall, these studies demonstrate the breadth of VNF placement research, addressing a spectrum of scenarios and goals. They commonly provide solutions for sufficient infrastructure resources but often overlook infeasible placement scenarios, highlighting the need for strategies that identify and resolve infeasibility.

B. VNF Placement Feasibility Restoration

The challenge of securing feasibility during VNF placement is critical to maintain network reliability and performance. Ensuring that sufficient resources are always there to handle the dynamic demands of the network, is a cornerstone of effective VNF management. In one of our earlier works [6], we have underlined the significance of identifying root causes for infeasibility in the underlying network infrastructure. This was achieved through an automatic feasibility restoration mechanism, pinpointing the primary reasons for infeasibility. This approach can adapt to dynamic changes in resource demands by suggesting rectifications based on elastic resource management. It showcases the essence of elasticity in resource management - an indispensable trait in networks with unpredictable demands.

Wang et al. in [16], proposed a dual-stage framework. In the pre-placement stage, the VNF placement is solved using the Grey Wolf Optimizer (GWO), which is guided by predictions on incoming workload. The goal here is to minimize resource consumption while adhering to delay constraints. The corrective stage, on the other hand, employs a greedier approach. Its primary function is to accommodate unpredicted SFC requests, yet doing so in a manner that keeps placement costs minimal. This work highlights the importance of adaptability in VNF placements, catering to both predicted and unpredicted network demands. However, this two-staged framework, while aiming for precision, might introduce unnecessary complexities. Multi-staged approaches often require more resources, both in terms of computation and administration, to be effectively implemented. Moreover, it relies on historical data that may not always capture the nature of real-time network changes. This might result in false positives or negatives, leading to either over-allocation or under-allocation of resources.

A Deep learning-assisted solution to feasibility restoration was presented by Pandey et al. in [17]. Their framework uses Deep Q-Networks (DQNs) to allocate VNFs at the network's edge. This solution is capable of renting resources from neighboring data centers when local resources fail to accommodate all VNF requests. However, relying on DQNs introduces a level of opacity in decision-making, often called the "black box" problem; interpreting and justifying the network's specific placement decisions might become challenging. Moreover, DQNs require significant amounts of data for training. In dynamic and evolving network environments, frequent retraining might be necessary to ensure the model remains relevant, which can be resource intensive.

C. Novelty of our Work in Comparison to the Literature

While the above methods offer promising solutions to the challenges of VNF placement and feasibility restoration, a comprehensive approach encompassing all facets of network planning is still lacking. Such a holistic method should cater to the entire lifecycle of service placement in an edge/cloud infrastructure, ensuring that the VNF placements are optimal, adaptable, and resilient to changing demands. The holistic approach has the primary advantage of providing an integrated view of the entire network. By considering all aspects of the network, from edge devices to core infrastructure, holistic planning is better positioned to synergize and optimize resources. Furthermore, flexibility is at the heart of holistic planning. It optimizes resource allocation by leveraging real-time network conditions and predictive insights. Rather than relying on reactive measures, it proactively identifies potential bottlenecks and reallocates resources to ensure peak network performance. Despite the current attempts and the implementation of a few existing techniques for VNF placement and feasibility restoration, we have encountered a deficiency of methods that take into consideration the complete range of a network planning tool: offline planning, online corrective actions, and semi-real time suggestions for locating and repairing potential infeasibilities during VNF placement.

Our VNF placement approach adeptly manages both planned and immediate network demands. For offline placement, we use a strategic, long-term approach to optimize VNF allocation, leveraging complete network information and demand projections for efficient initial setup or significant upgrades. This involves advanced computations to prepare the network for future demands. In contrast, our online VNF placement is dynamic, focusing on instant decisions in response to unforeseen network changes, using real-time data without future state knowledge. It employs rapid-response algorithms and feasibility restoration techniques for optimal VNF allocation amidst fluctuating conditions, using real-time analytics and adaptive heuristics for instant resource

reconfiguration. Our integrated offline and online strategies provide a robust and flexible solution for efficient network resource management, enabling operators to swiftly adapt to new demands and issues as they emerge.

Hence, our research endeavors to explore the problem at hand and establish versatile solution strategies for VNF placement and feasibility repair. We suggest precise and heuristic methods and assess both offline and online VNF allocation as well as suggestions for the SPs and InPs on how to accommodate more SFC requests in their infrastructures, if needed. Furthermore, our analysis focus on an edge/cloud interplay, which enables end-users to encounter reduced latency in an integrated edge and cloud infrastructure. This could lead in reducing the expenses (i.e., CapEx and OpEx) for the involved stakeholders, while concurrently upholding the quality of communication services.

III. System Model

In this section, we provide the modeling of the system that will be the input to the VNF placement and SFC planning algorithms. In particular, we model the physical infrastructure as a 3-tier network, consisting of the edge, transport and core (cloud) tiers, as shown in Fig. 1.

The physical network is represented as an undirected graph $\mathbb{G} = (\mathbb{H}, \mathbb{L})$, where $\mathbb{H}$ denotes the set of nodes and $\mathbb{L}$ the set of physical links. In VNF placement problems, the physical network is described as an undirected graph for simplicity and efficiency. This representation captures communication links' bi-directionality, providing a clear view of node connectivity without repetitious connections. An undirected graph facilitates algorithmic complexity and computation times, ensuring efficient and effective VNF placement. The nodes can be either servers $\mathbb{N} \subset \mathbb{H}$ or routers $\mathbb{W} \subset \mathbb{H}$, with $\mathbb{N} \cup \mathbb{W} = \mathbb{H}$. The servers are further classified based on their location as edge servers $\mathbb{N}_\mathbb{E} \subset \mathbb{N}$ and core servers $\mathbb{N}_\mathbb{C} \subset \mathbb{N}$, with $\mathbb{N}_\mathbb{E} \cup \mathbb{N}_\mathbb{C} = \mathbb{N}$. Similarly, the routers are separated into gateway routers $\mathbb{W}_\mathbb{Q} \subset \mathbb{W}$, which are the network's entry points from where end-users connect to their requested SFCs, and normal routers $\mathbb{W}_\mathbb{O} \subset \mathbb{W}$ that interconnect the three network tiers, with $\mathbb{W}_\mathbb{Q} \cup \mathbb{W}_\mathbb{O} = \mathbb{W}$. Finally, each server $n \in \mathbb{N}$ is attributed with a vector of available resources $R(n)$ (i.e., CPU, memory, storage), while every link $(u, v) \in L$ is characterized by a bandwidth capacity $b(u, v)$ and a propagation delay $d_p(u, v)$.

Regarding the SFCs, we assume that there is a set $\mathbb{S}$ of SFCs, representing different types of network services. Each SFC $S^i \subset \mathbb{S}$ consists of a number of K interconnected VNFs, such that $S^i = (s^i_0, s^i_1, s^i_2, s^i_K)$, with s^i_0 being the gateway router, $w^i_q \in \mathbb{W}_\mathbb{Q}$, from which a user requesting the SFC S^i is accessing the network. Following [18], we assume that the VNFs can be interconnected in various ways (e.g., linear or bifurcated). Each VNF s^i_k in an SFC S^i is characterized by a vector of resource demands $P(s^i_k)$ (i.e., CPU, memory, storage) and a maximum processing delay $d^{max}_{pr}(s^i_k)$ that need to be satisfied. The incurred processing delay depends on the capabilities of the server that will host the particular VNF $d_{pr}(s^i_k, n)$, $n \in N$, and it must hold that $d^{max}_{pr}(s^i_k) \leq d_{pr}(s^i_k, n)$. Naturally, the incurred processing delay of each VNF depends on whether the VNF is hosted on an edge or a core server. This is based on the fact that core servers are more powerful and can yield lower processing delays than edge ones. Additionally, each virtual link of an SFC that interconnects two VNFs, i.e., $(s^i_k, s^i_{k'})$ has a bandwidth requirement $b(s^i_k, s^i_{k'})$ and a maximum propagation delay $d_p(s^i_k, s^i_{k'})$ that need to be satisfied. Finally, each SFC requests a maximum end-to-end delay $d_{E2E}(S^i)$ that must be respected, which is the sum of the total processing and propagation delay noticed in all VNFs and virtual links.

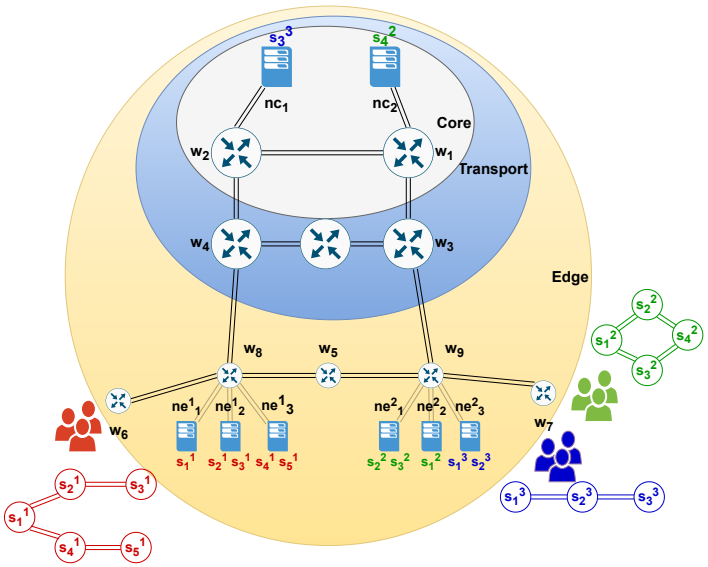

Fig. 1. An overview of a network infrastructure and some SFC instances.

Fig. 1 shows an example of the physical infrastructure and some requested SFCs. In particular, three different SFCs are considered, all having different topological characteristics (illustrated with red, blue, and green colors, respectively). The SFCs start at the gateway routers W_Q where the users are connected (i.e., w_6 for the red SFC and w_7 for the blue and green). The problem that we target in the first part of this paper is the following: considering the physical network of the InP and an estimation of the number of SFCs, their type and their expected entry point/gateway, find the most appropriate SFC planning or VNF placement solution. The solution should satisfy the interests of both the InP and the users.

IV. Offline and Online SFC Planning

In this section, we mathematically formulate the SFC planning problem described above. The ultimate goal is to find the optimal solution that minimizes the computational and communication resources used (as this constitutes the typical SFC's deployment cost [19]) and satisfies the user's SFC requirements under a number of constraints imposed by the InP. Our approach is designed to cater to the specific needs of both SPs and InPs, ensuring optimal resource utilization and compliance with user-defined constraints. SPs are primarily concerned with minimizing operational costs and maximizing service availability and quality, which translates to efficient resource usage and adherence to SLAs. In contrast, InPs optimize infrastructure utilization, balance load, and ensure long-term scalability and maintainability.

When considering where to place VNFs, both the network's core and edge provide distinct advantages. The core, known for its robust computational power, is a centralized hub for efficient processing-intensive tasks critical for latency-sensitive applications. This centralization allows for better resource aggregation and dynamic allocation and better adaptation to changing traffic patterns and service demands. Furthermore, through its redundant infrastructure, the core ensures higher resilience and maintains consistent Quality of Service (QoS) policies across various network segments. On the other hand, the placement of VNFs at the network edge significantly reduces latency by shortening the data travel distance, which is advantageous for real-time applications such as AR/VR and online gaming. Edge placement also promotes localized data processing for quick decision-making and effective traffic offloading, potentially reducing core congestion. Additionally, VNFs at the edge can be tailored to specific regional or local needs, ensuring personalized service delivery. Edge placements prioritize fast localized processing and low latency, whereas core placements prioritize robust computation and policy uniformity. Specific application requirements often determine the best option, and some scenarios may benefit from a hybrid placement technique; therefore, in this paper, we consider a strategic hybrid placement.

A. Offline ILP Formulation

The above problem can be formulated as an Integer Linear Programming (ILP); to this end, two binary decision variables are introduced, $x_n^{s_k^i}$ that is set equal to 1 if the VNF s_k^i is allocated on the server $n \in \mathbb{N}$ and $y_{uv}^{s_k^i s_{k'}^i}$ which is set equal to 1 if the virtual link between s_k^i and $s_{k'}^i$ is routed over the physical link $(u,v) \in \mathbb{L}$. Hence, the objective function can be expressed as follows:

$$\min_{x,y} \sum_{i=1}^{|S|} \Big(\sum_{s_k^i}^{S^i} \sum_{n}^{\mathbb{N}} M_n w_n x_n^{s_k^i} + \sum_{s_k^i}^{S^i} \sum_{s_{k'}^i}^{S^i} \sum_{(u,v)}^{\mathbb{L}} w_{uv} y_{uv}^{s_k^i s_{k'}^i} \Big). \quad (1)$$

The goal of this objective function is twofold. The first term tries to minimize the number of used edge servers by introducing the binary cost M_n. In particular, M_n takes the value of 1 if the server n is an edge server ($n \in \mathbb{N}_{\mathbb{E}}$) and 0 otherwise. This can be reasoned by the fact that cloud servers are much more powerful than their edge counterparts. Hence, to avoid using "expensive" edge resources and clogging the Edge of the network, the cloud servers will be prioritized, provided they respect the constraints of the SFC (i.e., in terms of E2E delay and resources). By incorporating M_n, the formulation prioritizes the use of cloud servers over edge servers, thereby minimizing the use of more "expensive" and scarce edge resources. The second term minimizes the communication cost by using as few links as possible to interconnect the different VNFs of an SFC. Reducing the number of used links directly contributes to minimizing the communication cost, which is a significant factor in network planning. Our ILP formulation is specifically designed to balance the trade-offs between minimizing computational and communication resources, a typical objective for SPs and ensuring high-quality service delivery within the constraints imposed by InPs.

Moreover, w_n and w_{uv} are the weights network operators can adjust to prioritize using server capacities or link routing based on their specific requirements and preferences. This added layer of flexibility ensures that the objective can be adapted to accommodate a wide range of network conditions and operational strategies. To simplify the presentation of the approach, and without hampering the generalizability of the objective function, in our implementation we assign equal weights $w_n = w_{uv} = 1, \forall n \in \mathbb{N}, \forall (u,v) \in \mathbb{L}$. The objective function is minimized subject to the constraints that follow:

$$d_{Total}(S^i) \leq d_{E2E}(S^i), \forall S^i \in \mathbb{S}, \quad (2)$$

$$\sum_{n \in \mathbb{N}} d_{pr}(s_k^i, n) x_n^{s_k^i} \leq d_{pr}^{max}(s_k^i), \forall s_k^i \in S^i, \forall S^i \in \mathbb{S}, \quad (3)$$

$$\sum_{(u,v) \in \mathbb{L}} d_p(u,v) y_{uv}^{s_k^i s_{k'}^i} \leq d_p(s_k^i, s_q^i), \forall s_{k|k'}^i, \forall S^i \in \mathbb{S}. \quad (4)$$

First, we need to guarantee that the total experienced delay of an SFC, $d_{Total}(S^i)$, is less than the E2E delay requirement (constraint 2), where $d_{Total}(S^i) = \sum_{s_k^i \in S^i} \sum_{n \in \mathbb{N}} d_{pr}(s_k^i, n) x_n^{s_k^i} + \sum_{s_k^i \in S^i} \sum_{s_{k'}^i \in S^i} \sum_{(u,v) \in \mathbb{L}} d_p(u,v) y_{uv}^{s_k^i s_{k'}^i}$. Furthermore, the individual processing delay requirements for each VNF (constraint 3) and the individual propagation delay requirement of each virtual link of an SFC (constraint 4) should also be ensured. It is worth noting that strict requirements for processing delay will lead to the allocation of the VNF at the core of the network, while strict propagation delay requirements will guide the solution to place the involved VNFs on the same or adjacent servers. The next set of constraints assures that the computational (constraint 5) and communication (constraint 6) resources are not oversubscribed:

$$\sum_{i=1}^{|S|} \sum_{s_k^i \in S^i} P(s_k^i) x_n^{s_k^i} \leq R(n), \forall n \in \mathbb{N}, \quad (5)$$

$$\sum_{i=1}^{|S|} \sum_{s_k^i \in S^i} \sum_{s_{k'}^i \in S^i} b(s_k^i s_{k'}^i)(y_{uv}^{s_k^i s_{k'}^i} + y_{vu}^{s_k^i s_{k'}^i}) \leq b(u,v), \quad \forall (u,v) \in \mathbb{L}. \quad (6)$$

The flow conservation constraint (7) ensures the interconnection of the VNFs by performing the routing between the source and destination of each virtual link. This constraint is defined for each virtual link independently, allowing it to accommodate various SFC topologies, including linear, branched, and more complex non-linear structures. This constraint ensures that the ingress and egress flows are appropriately balanced, regardless of the overall SFC topology:

$$\sum_{v \in \mathbb{H}} (y_{uv}^{s_k^i s_{k'}^i} - y_{vu}^{s_k^i s_{k'}^i}) = (x_u^{s_k^i} - x_u^{s_{k'}^i}), \quad \forall u \in \mathbb{H}, \forall s_{k|k'}^i \in S^i, \forall S^i \in \mathbb{S}. \quad (7)$$

Finally, we need to ensure that each VNF will be placed at only one node (constraint 8), provided that the specific node is a server (constraint 9), except the first VNF of an SFC that simply denotes the entry gateway of the user requesting the SFC (constraint 10):

$$\sum_{h \in \mathbb{H}} x_h^{s_k^i} = 1, \forall s_k^i \in S^i, \forall S^i \in \mathbb{S}, \tag{8}$$

$$\sum_{n \in \mathbb{N}} x_n^{s_k^i} = 1, \forall s_k^i \in S^i \setminus \{s_0^i\}, \forall S^i \in \mathbb{S}, \tag{9}$$

$$x_{w_q^i}^{s_0^i} = 1, \forall S^i \in \mathbb{S}. \tag{10}$$

The above formulation will give the global optimal solution for all SFCs expected to be serviced by the network. Hence, given all the SFCs' requirements and the current state of the network, the ILP solver decides the optimal allocation and interconnection of VNF instances across the underlying network infrastructure based on the objective in Eq. 1.

However, since the problem of interest is an NP-hard problem [20], the execution time exponentially increases with the size of the network. Even though long execution times can be acceptable for offline network planning (e.g., "dimension the network for the next day/week/month"), in order to avoid any computational intractable situation, we additionally propose two offline heuristic approaches, namely SeqSort and SeqBiased. Our SeqSort and SeqBiased heuristics were designed to provide practical, near-optimal solutions in less time than the global ILP method. These heuristics' core philosophy is to simplify the problem by breaking it down into manageable parts, processing them sequentially, or prioritizing based on specific criteria, thereby avoiding the exhaustive search space that defines the problem's NP-hard nature.

1) Sequential Sort (SeqSort): SeqSort is a sequential heuristic in which the ILP method is executed for a single SFC at a time, and the state of the network resources is updated accordingly. Instead of finding the allocation for all the SFCs in batches, as in the global offline ILP method, SeqSort first sorts the SFC requests in descending order based on the weighted sum of their computational demands (i.e., CPU, memory and storage). For example, given a service request S^i with the following demands, 4 CPU cores, $4000MB$ of RAM, and $400GB$ of storage, $\sum_{s_k^i \in S^i} P(s_k^i) = [4, 4000, 400]$, a weighted average of the demands is calculated as follows: $P_{av}(S^i) = \lceil (P[0] + P[1]/1000 + P[2]/100)/3 \rceil$. Thus, in this case, $P_{av} = \lceil (4 + 4000/1000 + 400/100)/3 \rceil = 4$. The purpose of using this weighted average is to prioritize SFC requests with higher computational demands. This approach handles more resource-intensive requests earlier in the sequence, increasing resource utilization and reducing potential resource allocation bottlenecks. Since SeqSort is not dealing with all the SFCs at the same time, it provides suboptimal solutions. However, SeqSort is still an offline solution since it requires the knowledge of all future SFCs that are expected to come. The pseudocode of SeqSort is given in Algorithm 1.

The first step involves calculating $P_{av}(S^i)$ for each SFC, which is done in $O(|S|)$ time where S is the number of SFCs. Sorting the SFCs based on $P_{av}(S^i)$ takes $O(|S|log|S|)$,

Algorithm 1: SeqSort

Inputs: $\mathbb{G}, \mathbb{S}$
Output: Total Cost, Total Execution Time, Acceptance Ratio

```
for (i = 0; i < |S|; i++) do
    Calculate P_av(S^i);
end
SortedSFCs ← Sort SFCs in descending order based on P_av(S^i);
TotExecTime = 0, TotObjCost = 0, Blocking = 0;
for (i = 0; i < SortedSFCs.size; i++) do
    Allocate S^i using an ILP solver;
    TotExecTime += S^i.ExecTime;
    TotObjCost += S^i.ObjCost;
    if S^i.allocated then
        Reserve resources for S^i;
    else
        Blocking ++;
    end
end
AcceptanceRatio = 1 − Blocking/|S|;
```

considering a standard sorting algorithm. The main loop iterates over each SFC, in which a single SFC is allocated in $O(|S|C_{solver})$, where C_{solver} is the ILP solver complexity for allocating a single SFC. In our implementation, we use Gurobi's ILP solver [21] which solves the problem using the branch-and-bound method, complemented with heuristics and reductions to speed up the practical running time. The computational complexity of this approach is still ultimately exponential. Therefore, the SeqSort Algorithm takes $O(|S| + |S|log|S| + |S|C_{solver})$ time to allocate all SFCs presented in the problem instance. The computational complexity of the SeqSort algorithm is dominated by the single SFC allocation step, which nonetheless is significantly less complex compared to the ILP problem for all SFCs. The other steps contribute a linear or log-linear component to the overall complexity.

2) Sequential Biased (SeqBiased): This one is also a sequential heuristic, as it allocates one SFC at a time, but unlike SeqSort, it does not resort to the ILP method. Computational concerns largely drove the decision not to utilize the ILP method in SeqBiased. ILP, although robust and optimal, can be computationally expensive, mainly when applied to large-scale and dynamic problems like the one we address. Our goal with SeqBiased was to present a heuristic that's efficient in terms of solution quality and computationally tractable, ensuring that it is practical for real-world deployment scenarios. Refraining from the ILP method, we could expedite the solution process and provide answers within reasonable timeframes, especially suitable for real-time or near-real-time demands.

First, it calculates the weighted sum of the computational resource requirements of each SFC $P_{av}(S^i)$ as before and sorts them in descending order. Secondly, the shortest paths between the SFCs' gateways and all the servers are determined, using the well-known Dijkstra algorithm. Then, a weighted sum $Z_{sum}(n, i)$ is calculated for each server based on its

average availability of resources $R_{av}(n)$; $R_{av}(n) = \lceil (R[0] + R[1]/1000 + R[2]/100)/3 \rceil$, $R_{av}(n)$ is calculated similarly to the $P_{av}(S^i)$, and the distance $d(i)$ from the gateway that the SFC S^i is coming from. The distance $d(i)$ is the sum of the links' propagation delay connecting the gateway w_q^i to server n. Thus, the servers' weighted sum is calculated as follows: $Z_{sum}(n,i) = a_1 * d(i) + a_2 * \lceil (1/R_{av}(n)) * \sigma \rceil$, where σ is a scaling factor and a_1, a_2 are priority weights. In our experiments, we set $a_1 = a_2 = 1$ and $\sigma = 10$ to prioritize servers closer to the user. Then, the servers are sorted in descending order. Finally, the VNFs of an SFC are allocated to the ordered servers, favoring those that are closer to the SFC's gateway and with enough available resources. The pseudocode of SeqBiased is given in Algorithm 2.

Algorithm 2: SeqBiased

Inputs: $\mathbb{G}, \mathbb{S}$
Output: Total Cost, Total Execution Time, Acceptance Ratio

for $(i = 0;\ i < |S|;\ i++)$ **do**
  Calculate $P_{av}(S^i)$;
end
$SortedSFCs \leftarrow$ Sort SFCs in descending order based on $P_{av}(S^i)$;
Execute Dijkstra's Algorithm;
for $(i = 0;\ i < |W_Q|;\ i++)$ **do**
  for $(n = 0;\ n < |N|;\ n++)$ **do**
    Calculate $Z_{sum}(n,i)$;
  end
  $SortedServers_i \leftarrow$ Sort servers in descending order based on $Z_{sum}(n,i)$;
end
$TotExecTime = 0, TotObjCost = 0, Blocking = 0$;
for $(i = 0;\ i < SortedSFCs.size;\ i++)$ **do**
  Allocate S^i in set $SortedServers_i$;
  $TotExecTime\ +=\ S^i.ExecTime$;
  $TotObjCost\ +=\ S^i.ObjCost$;
  if $S^i.allocated$ **then**
    Reserve resources for S^i;
  else
    $Blocking$ ++;
  end
end
$AcceptanceRatio = 1 - Blocking/|S|$;

The initial steps for calculating $P_{av}(S^i)$ and sorting the SFCs are similar to SeqSort, with a complexity of $O(|S|)$ and $O(|S|log|S|)$, respectively. Running Dijkstra's algorithm adds a complexity of $O(|E|log|N|)$, where $|E|$ is the number of edges, and N is the number of nodes in the network graph. The nested loops for calculating $Z_{sum}(n,i)$ and sorting servers have a complexity of $O(|S||N|log|N|)$. Therefore, the SeqBiased Algorithm takes $O(|S| + |S|log|S| + |E|log|N| + |S||N|log|N|)$ time to allocate all SFCs presented in the problem instance. The SeqBiased algorithm, therefore, has a complexity mainly influenced by Dijkstra's algorithm and the servers' sorting steps. Since some of these terms might be dominant over others depending on the specific values of $|S|$, N, and $|E|$, the total complexity will be influenced by the most significant terms in practical scenarios.

In our context, offline heuristics methods primarily use predefined rules based on network state and expected workloads. These criteria consider the network's current and projected bandwidth usage, computational capacities at different network nodes, expected end-to-end latency, and the requirements for specific service functions. Offline heuristics aim to use the predictions of SFCs' requests to strategically place it in locations that offer optimal performance while balancing the trade-offs between computational overhead and network congestion by evaluating the network's existing conditions against the expected demand.

By definition, offline approaches operate based on prior knowledge or estimations of incoming SFC requests, and they frequently run batch optimization processes based on this knowledge. When the estimations are correct, this method results in efficient resource utilization, as the algorithm can examine the entire set of requests before deciding. The primary limitation of offline approaches, however, is their rigidity; the system commits to those allocations once the process is complete, meaning there is no flexibility to adjust in real-time in case of unexpected SFC request arrivals or inaccurate estimations. The entire optimization process would have to be re-run, which is not possible in a real-time scenario, especially given the computational time required for optimization. On the other hand, online methods, that make decisions as requests arrive, are adaptable to changing conditions or inaccuracies in demand prediction. This adaptability is due to their inherent design, which allows them to make decisions without prior knowledge of the entire request sequence, allowing for flexibility in changing scenarios.

B. Online SFC Planning

The above approaches work well in an offline setting that assumes that all the SFCs that will co-exist on the physical infrastructure at the same time, are known in advance. This makes them good candidates for network planning, as long as the estimation of the number of incoming SFCs and the state of the physical network are accurate. However, as explained, if the estimations are inaccurate (e.g., more SFC requests are submitted from possibly different gateways), the offline approaches cannot provide alternative solutions in real-time. Accordingly, in this part of the section we present several online approaches that can independently place each incoming SFC while also leveraging the results of the global ILP offline solution to help the network operator take corrective measures. The online algorithms will have to take into consideration the arrival and the departure times of the SFCs, which could depend on the type of service requested and the location of the end-users. Moreover, following the allocation or de-allocation of any SFC, the state of network resources is updated in real-time within the ILP model. This dynamic update ensures the model constantly works with the most recent network conditions, allowing for accurate placement decisions. In contrast to the offline approach, our online SFC

planning strategies are designed for real-time adaptability, a crucial requirement for SPs in rapidly changing network environments. Online strategies consider the arrival and departure times of SFCs, which vary based on service type and user location, directly impacting SPs' ability to deliver continuous and reliable services.

1) Pre-allocation Method: Pre-allocation refers to setting aside resources for future use based on predicted needs. The idea is to have resources ready to use when demands come in, which can reduce real-time computational overhead and accelerate service provisioning. In this method, a number of preallocated solutions produced during the offline execution of the global ILP can be used. In particular, during the network planning phase, the network operator can decide the number, type, and location of the preallocated solutions beforehand, based on the expected network services to be deployed and pass them as input to the ILP model. Following, as service requests arrive to the network in real time, the method allocates them using the preallocated solutions. Hence, no placement algorithm is executed in run time. Once the network runs out of preallocated solutions, it will start blocking the incoming requests. This method needs to wait for preallocated solutions to become free again before admitting new SFCs.

2) Pre-allocation plus online ILP Method: Similar to the previous method, this approach allocates the incoming requests using the preallocated solutions. However, it does not block the request immediately once the network runs out of preallocated solutions. Instead, it finds a new solution by formulating a new ILP problem for allocating only the blocked SFC using the following objective function:

$$\min_{x,y}\Big(\sum_{s_k^i\in S^i}\sum_{n\in\mathbb{N}} M_n x_n^{s_k^i} + \sum_{s_k^i\in S^i}\sum_{s_{k'}^i\in S^i}\sum_{(u,v)\in\mathbb{L}} y_{uv}^{s_k^i s_{k'}^i}\Big). \quad (11)$$

This way, the network operator can find a remedy solution if the network planning during the offline pre-allocations stage is inaccurate. In detail, this formulation considers available and occupied resources, the specific requirements of the blocked SFC, and the existing network topology, to form the objective function. Integral constraints include resource availability, maintaining the SFC's inherent sequence, and ensuring that latency limits are not exceeded. This approach aims to achieve optimal resource allocation for the blocked request by carefully considering these factors to improve system efficiency and acceptance rate.

3) Pre-allocation plus online Greedy Method: This method is similar to the preallocation plus online ILP method. However, here we find a new solution for the blocked service request by using the SeqBiased heuristic instead of using a single-SFC ILP formulation. The choice to employ the SeqBiased heuristic in the context of the pre-allocation plus online greedy method, as opposed to resorting to a single-SFC ILP formulation, is based on several key considerations. First, the computational efficiency of the SeqBiased heuristic is notably superior in real-time scenarios compared to the computationally intensive ILP formulations. Such efficiency is pivotal for immediate, on-the-fly adjustments. Secondly, the inherent design of SeqBiased is agile, allowing it to adjust to dynamic changes in the network state adeptly and can offer feasible solutions even in situations where the strict ILP might struggle. Lastly, whereas an ILP approach can introduce overheads, particularly for individual SFC requests, SeqBiased efficiently mitigates these overheads due to its heuristic nature, making it more fitting for real-time single SFC allocations.

V. Infeasibility Restoration for SFC Planning

The global offline ILP is an efficient algorithm for finding the optimal solution for a batch of SFCs. However, if the available physical resources of the InP are not enough or if a constraint cannot be satisfied, the ILP becomes infeasible. This means that either all the SFCs will find a suitable placement or none will be allocated. This problem can be alleviated with the two offline heuristic approaches proposed that solve the problem for one SFC at a time. After applying either of them, only the excessive SFCs will be blocked due to resource limitations in the infrastructure. Nonetheless, an InP naturally needs to have a better insight into the cause of infeasibility and plan their network resource allocation accordingly. Often, infeasibilities in network resource allocation can arise from various factors such as dynamic traffic patterns that might suddenly surpass the anticipated loads, physical hardware constraints and hardware malfunctions, software misconfigurations or failures, and external factors like network attacks or large-scale outages. These variables can converge in complex ways for InPs, making it difficult to pinpoint the precise reasons for an allocation's impracticality. InPs need a systematic approach to diagnosing and understanding these obstacles to optimize resource allocation.

Thus, in this section, we propose infeasible restoration techniques that will allow the global offline ILP to not only detect infeasibilities, but also provide suggestions to the InPs on where more resources (such as CPU, Memory, Storage and bandwidth) should be added in the network to restore the infeasibility. To do so, the additional resources should be associated with a cost, meaning that the least expensive resources will be added to the network first, reducing this way the additional capital and operational expenses of the provider.

For this reason, two approaches are introduced for the infeasibility analysis. The first approach is called Irreducible Infeasible Set (IIS) Repair and leverages the information of the unsatisfied constraints that cause the infeasibility. The second approach is called Minimum Cost Redesign and it converts the infeasibility analysis problem to an optimization problem using elastic variables. After determining the sources of infeasibilities, our algorithms automatically recommend a remedy by finding out the repair with the best cost.

A. Irreducible Infeasible Set

One of the standard methodologies for infeasibility analysis is identifying the constraints' Irreducible Infeasible Set (IIS). IIS is defined as a subset of the model constraints and variable boundaries that cannot be jointly met [22]. However, the removal of any member of the IIS could make the remaining members of the IIS jointly satisfiable. Thus, IISes can be used in identifying the grounds of infeasibility and are

essential components in infeasibility analysis. The size of an IIS depends on the number of constraints in the system. For an infeasible model with many constraints, IIS generally contains many components, making it challenging to identify the source of infeasibility accurately.

Typically, when we have an infeasibility, our goal is to determine which changes should be made to restore the feasibility. However, a challenging aspect is that not all of the constraints can be modified. For example, adding a new and long fiber to interconnect two data centers may not be realistic in order to solve an infeasibility caused by limited communication resources. Similarly, the addition of a new edge datacenter can solve infeasibilities emerging from the lack of computational resources but may be too expensive for a network operator. Therefore, it is vital to identify those constraints in the IIS that must be met (i.e., unmodifiable constraints), from those that might perhaps be relaxed (i.e., modifiable constraints). Subsequently, the analytical efforts can be focused on the modifiable constraints only. In our case, modifiable constraints refer to adjustable parameters, such as resource capacities. On the other hand, unmodifiable constraints represent fixed conditions, rules, or requirements that must be strictly satisfied for a solution to be feasible, such as user location, strict limitations (i.e., E2E delay) or regulatory requirements.

Furthermore, another challenge is that the IIS is not unique. For example, an infeasible model may have more than one IISes [23]. This is because after every execution, a different set of constraints may be identified as an IIS. Therefore, the removal of any member of a specific IIS is not necessarily enough to fix the infeasibility of the model; we need to relax at least one member from each IIS to restore the feasibility.

To better understand the concept of an IIS, let C define a set of jointly infeasible constraints, then the subset $I \subseteq C$ is an IIS of C if and only if I is infeasible and when relaxing any member of I makes the problem feasible. Moreover, I is a Minimum-Cardinality IIS Set Cover ("Min. IIS Cover") of C iff $C \backslash I$ is feasible, and the addition of any $c \subseteq I$ to $C \backslash I$ makes it infeasible. Every Min. IIS Cover is a hitting set of all IISes, i.e., intersects with each IIS. Eliminating an IIS from C, i.e., relaxing one of its members, may not make it feasible if C has multiple IISes. However, removing a Min. IIS Cover from C, i.e., relaxing all its members, is guaranteed to make C feasible [22].

Thus, our goal is to identify the IIS and relax the minimum set of constraints that make the model feasible. The relaxing of the said constraints will be translated into the addition of resources to the network infrastructure that will help the provider make a better planning according to the expected demand. Obviously, this addition should incur the minimum cost for the provider and this will be the goal of the infeasibility restoration techniques presented below.

B. IIS Repair

The IIS repair approach leverages the information provided by the IIS. Several methods for identifying a single IIS of an infeasible model can be found in the literature [22]. However, as stated above, an IIS is not unique, and an infeasible problem can have many IISes. Hence, to find the optimal repair, we need to uncover all the IISes. Unfortunately, finding all IISes is an NP-Hard problem [24]. The general idea of IIS repair is to pursue find-and-repair cycles. In each cycle, the model is solved, and an IIS is identified and corrected. Then, the determined IISes are analyzed to select which constraint to relax and add it to the IIS Cover. This is repeated until the model becomes feasible. We propose two algorithms for the IIS repair, namely the CostBased and the ShuffleFilter.

1) CostBased: A graphical illustration of the CostBased method is shown in Fig. 2. CostBased is a sequential method that follows a find-and-repair cycle. A single IIS is identified and repaired in each cycle. After pinpointing an IIS, CostBased identifies the constraints that can be altered, in our case the network resources capacity constraints. Then it goes through the modifiable constraints of the IIS and assigns a cost to each one. CostBased relaxes the constraint with the lowest cost and then adds it to the IIS Cover. Following, it inspects if the model is feasible or not. If the model is still infeasible, it reiterates the process. For example in Fig. 2, when solving the ILP model for the first time we find that the first IIS detected returns three constraints ($c1$, $c2$, and $c3$). By relaxing constraint $c1$, adding it into the current IIS Cover set and resolving the problem, a new IIS is produced, since two constraints are still not satisfied (namely $c2$ and $c4$). By relaxing constraint c4, adding it into the IIS Cover and re-executing the ILP model, we end up with a third IIS that still has two unsatisfiable constraints ($c2$, and $c5$). The process continues until the problem becomes feasible.

The constraint repair cost reflects the amount of additional resources needed to relax that specific constraint. The modifiable constraints correspond to the communication and computation resources, such as CPU, memory, storage and bandwidth. Each resource type is given a weight so that the cost of different resources is comparable. The cost is calculated as follows: the number of extra resources are multiplied by their correspondent weight, where W_{CPU}, W_{Mem}, $W_{Storage}$, and W_{BW} are the CPU, memory, storage, and bandwidth weights, respectively. We set the weights as following: $W_{CPU} = 1$, $W_{Mem} = 1/1000$, $W_{Storage} = 1/100$, and $W_{BW} = 1$. Although CostBased is a sequential technique that employs a quick find-and-repair procedure, the provided repairs may not be the path to the optimal feasible solution, i.e., the solution with the least amount of constraints relaxations, and thus cost. The pseudocode is given in Algorithm 3.

2) ShuffleFilter: To reduce the above problem, we introduce ShuffleFilter, another IIS repair sequential method that also follows a find-and-repair procedure. However, here in each cycle we attempt to find more than one IISes; we call them initial IISes. Fig. 3 shows an example and the flow chart of the ShuffleFilter algorithm. ShuffleFilter goes through the modifiable constraints of the IIS and assigns a cost and priority to each constraint. The priority reflects the number of occurrences of a particular constraint in the initial IISes. If a specific constraint is included frequently in the initial IISes, then it has a higher probability of being included in other uncovered IISes as well. Thus relaxing that constraint will yield a higher chance of repairing more IISes, compared to

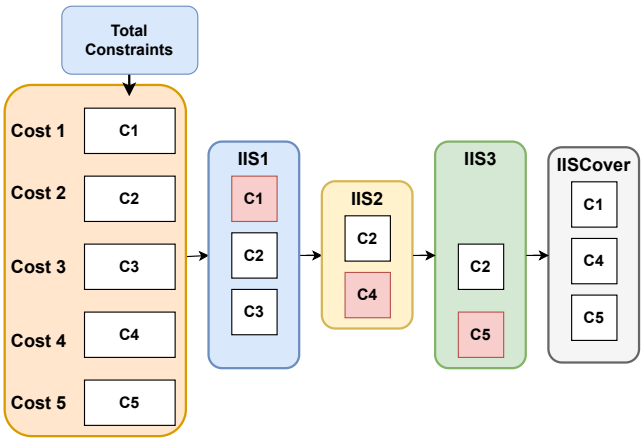

Fig. 2. An overview of the CostBased IIS Repair method.

Algorithm 3: CostBased IIS Repair

Inputs: Infeasible ILP model
Output: IIS Cover, Additional Resource Cost

```
Initialize IIS_Cover as empty set;
Initialize AdditionalResourceCost as 0;
while Model is Infeasible do
    Solve Model to identify an IIS;
    if No IIS found then
        Exit loop;
    end
    Initialize MinCost as infinity;
    Initialize ConstraintToRelax as null;
    foreach constraint in IIS do
        if constraint is modifiable then
            Calculate cost of relaxing the constraint;
            if cost < MinCost then
                MinCost = cost;
                ConstraintToRelax = constraint;
            end
        end
    end
    Relax ConstraintToRelax;
    Add ConstraintToRelax to IIS_Cover;
    Update AdditionalResourceCost by adding
     MinCost;
    Update Model with relaxed constraints;
    Re-check feasibility of the Model;
end
```

relaxing other constraints with lower priority. Still, that high priority constraint may lead to a shorter path to feasibility but not necessarily the one with the best repair cost. For instance, in Fig. 3, during the first cycle and for the three IISes identified, we notice that constraint $c2$ appears in all of them. Thus, this particular constraint will have the highest priority to be relaxed. If more than one constraints have the same priority, then the repair cost is used to decide which constraint to select. If two or more constraints share the exact cost and priority, the method makes a random selection. To find the initial IISes, ShuffleFilter relies on a shuffle method and multiprocessing, while it uses a hybrid Additive-Deletion filter to isolate a single IIS [22]. The hybrid additive-deletion filter allowed for more accurate identification of IISes compared to other filtering methods. It integrated the strengths of both additive and deletion methods, making the process less prone to false positives or missed IIS identifications. Due to its dual nature, this hybrid method managed to skip some iterations that would be needed if the two methods were used independently, thereby accelerating the IIS identification process. Also, one of the stimulating reasons to use the hybrid additive-deletion filter was its compatibility with the shuffle method.

The primary goal of the shuffle method is to diversify the order in which constraints are analyzed, potentially leading to the discovery of different IISes. By rearranging the order of constraints, the algorithm can expose different infeasible combinations, effectively uncovering more IISes than a static order would allow. In detail, finding the initial IISes is done as follows: if a given infeasible model has more than one IISes, we attempt to uncover more than one IIS using the Additive-Deletion filter by shuffling the order of the constraints and applying the filter multiple times. A generic notation $i \times j$ can be used to denote the number of parallel processes (i) and the number of times each process shuffles the model constraints and applies the Additive-Deletion filter (j). In our setting, ShuffleFilter runs eight processes in parallel in each cycle ($i = 8$), and each process will shuffle the model constraints and apply the Additive-Deletion filter eight times ($j = 8$). So, ShuffleFilter shuffles the constraint and applies the Additive-Deletion filter 64 times for each cycle. The more initial IISes found, the better the selection decision of which constraint to relax; however, the number of initial IISes found in each cycle is not fixed. Compared to CostBased, ShuffleFilter employs a better but slower find-and-repair procedure since, at each cycle, multiple IISes will be found.

The decision to use eight parallel processes in each cycle of the ShuffleFilter was primarily influenced by several considerations. Our primary experimental setup was based on a multi-core architecture that efficiently supported concurrent processing on eight cores without considerable resource contention. Additionally, in preliminary trials with varying numbers of processes, we observed that using more than eight parallel processes led to significant overhead, causing diminishing returns in performance. Conversely, deploying fewer processes did not effectively harness the available computational resources. Thus, to maintain consistency across various tests and comparisons, we found that eight processes provided an optimal balance between performance and resource utilization.

We finally provide an illustrative example of an IIS repair in VNF placement to clarify how it works. Consider a scenario of optimizing VNF placement within a cloud data center equipped with 10 servers. The capacities for each server are as follows: $16CPU$ cores, $32GB$ of RAM, $1TB$ of storage, and $1Gbps$ bandwidth. For this data center, we have the following VNF demands from three customers: A) 2 VNFs each requiring $6CPU$ cores, $10GB$ RAM, $100GB$ storage, and 300 Mbps bandwidth, B) 3 VNFs each requiring $4CPU$ cores, $8GB$ RAM, $150GB$ storage, and $250Mbps$ bandwidth

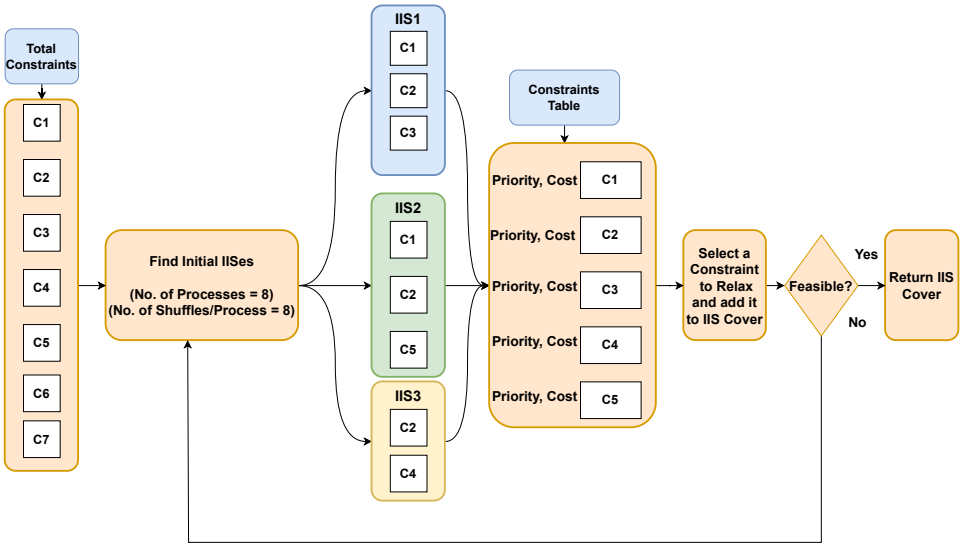

Fig. 3. An overview of the ShuffleFilter IIS Repair algorithm.

and C) 1 VNF requiring $8CPU$ cores, $12GB$ RAM, $300GB$ storage, and $500Mbps$ bandwidth. When attempting to fulfill these demands against the available servers' capacities, certain infeasible placements are encountered. Utilizing IIS, conflicts are discovered when deploying VNFs for customers A and B on servers 1 and 2, due to the excessive CPU requirements.

With the CostBased approach, the initial deployment attempt for customers A and B's VNFs on servers 1 and 2 results in constraints $c1$, $c2$, and $c3$, clashing. By referring to the CostBased illustration in Fig. 2, after addressing constraint $c1$ by upgrading the capacity of either server 1 or server 2, new conflicts $c2$ and $c4$ are identified. The method continues in this iterative fashion until a feasible solution emerges.

Using ShuffleFilter, the first cycle identifies three IISes spread across servers 1, 2, and 3. It is determined that constraint $c2$ is common across all identified sets. In the ShuffleFilter illustration (Fig. 3), after analyzing the three IISes from the first cycle, it becomes evident that constraint $c2$, associated with memory, consistently emerges. Thus, adjustments are made to the servers to alleviate this constraint. This example illustrates the practical challenges associated with VNF placements and how methodologies like CostBased and ShuffleFilter can facilitate feasible solutions. The pseudocode for ShuffleFilter is given in Algorithm 4.

Algorithm 4: ShuffleFilter IIS Repair

Inputs: Infeasible ILP model, Number of Shuffle Cycles $nCycles$, Number of Shuffles per Cycle $nShuffles$
Output: IIS Cover, Additional Resource Cost

```
Initialize IIS_Cover as empty set;
Initialize AdditionalResourceCost as 0;
while Model is Infeasible do
    Initialize CombinedIIS as empty set;
    for cycle = 1 to nCycles do
        for shuffle = 1 to nShuffles do
            Shuffle constraints of the model;
            Apply Additive-Deletion filter to find an IIS;
            Add found IIS to CombinedIIS;
        end
    end
    Analyze CombinedIIS to determine priorities of constraints;
    Select the constraint with highest priority and lowest cost;
    Relax the selected constraint;
    Add the relaxed constraint to IIS_Cover;
    Update AdditionalResourceCost by adding the cost of relaxation;
    Update Model with relaxed constraints;
    Re-check feasibility of the Model;
end
```

C. Elastic Heuristic

The last infeasibility restoration model proposed is the elastic heuristic. Elastic programming or elastic filter [22] creates a feasibility relaxation model by converting the infeasibility analysis problem into an optimization problem by using elastic variables. Specifically, this method adds elastic non-negative variables in each constraint, allowing them to be violated, while at the same time a penalty is added to the objective function that reflects a cost of the violation. In other words, the optimal solution of the feasibility relaxation model minimizes the sum of the weighted magnitudes of the constraint violations, which in turn can be translated into minimizing the number of violations needed in the constraints causing the infeasibilities in the model.

Inspired by the elastic programming, we propose the ElasticHeuristic method. The primary objective here is to offer a systematic method to transition from an infeasible solution space to a feasible one. Additionally, instead of a binary

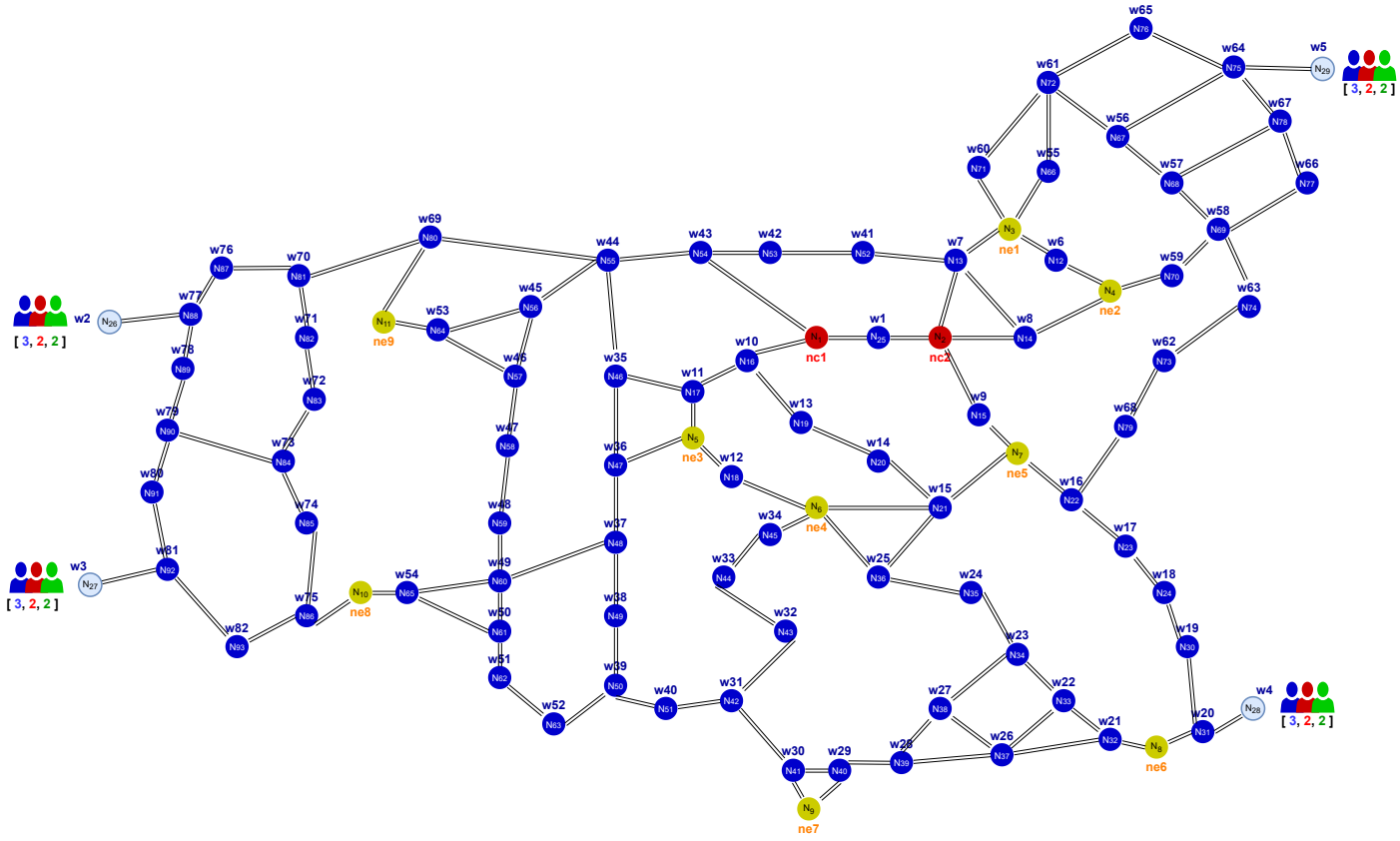

Fig. 4. Metropolitan Network

feasible-infeasible result, the transformed approach reveals where and to what extent constraints can be relaxed to allow all incoming SFC requests to be deployed.

To formulate the elastic ILP, we introduce the variable e_i as the elastic non-negative variable added to constraint i, and p_i the penalty for violating this constraint. Thus, the *modifiable* constraints are altered as follows:

- Constraints of the form $f_i(x) \leq b_i$ become $f_i(x) - e_i \leq b_i$
- Constraints of the form $f_i(x) \geq b_i$ become $f_i(x) + e_i \geq b_i$
- Constraints of the form $f_i(x) = b_i$ become $f_i(x) + e_i^+ - e_i^- = b_i$

Constraints that *cannot be modified* retain their original form. The objective is to minimize $Z = \sum_i (p_i e_i + p_i e_i^+ + p_i e_i^-)$, i.e., minimize the sum of the weighted magnitudes of the constraint violations, which leads to an automatic relaxation only of the constraints that cause the infeasibilities. Elastic variables only obtain non-zero values when the constraint is violated; therefore, the corresponding penalty value in the objective function should be considerably big. Based on that, we assume that the infeasibilities arise due to conflicts in constraints related to network resources, encompassing both computational and communication resources. It's worth noting that these constraints are the modifiable ones, while others, such as the end-to-end delay, cannot be altered. To distinguish between modifiable and unmodifiable constraints, we employ penalties. If a constraint is violated, the violation has a magnitude: how much the solution deviates from the constraint's requirement. This magnitude is then multiplied (or "weighted") by a penalty factor, which signifies the "cost" or "penalty" of violating that specific constraint. The bigger the violation, the bigger the penalty. The pseudocode for Elastic Heuristic is given in Algorithm 5. Therefore, the ILP objective (1) becomes:

$$\min_{x,y} \sum_{i=1}^{|S|} \left(\sum_{s_k^i \in S^i} \sum_{n \in \mathbb{N}} M_n x_n^{s_k^i} + \sum_{s_k^i \in S^i} \sum_{s_{k'}^i \in S^i} \sum_{(u,v) \in \mathbb{L}} y_{uv}^{s_k^i s_{k'}^i} \right) + Z.$$

D. Challenges and Limitations

We encountered several difficulties when transitioning from an infeasibility analysis to an optimization problem using the minimum cost redesign approach and elastic variables. It was critical to determine which constraints to relax with elastic variables. Constraints deemed essential to the problem formulation were left unchanged, while others were given some freedom. Finding this fine line while also ensuring the model's integrity was hard. Moreover, imposing proper penalties for constraint violations was crucial. Balancing leniency and severity needed iterative tuning and validation to ensure that optimization targets minimal constraint violations without eliminating feasible solutions. Also, altering constraints, especially equality constraints, was complex. For an equation like $f_i(x) = b_i$, introducing two elastic variables (e_i^+ and e_i^-) increased the problem's dimensionality. On the benefits side, because of the reformulation, instead of a mere feasible-infeasible outcome, now the solution offers insights into which

Algorithm 5: Elastic Heuristic

Inputs: Infeasible ILP model, Penalty weights P
Output: Feasible solution with minimized constraint violations

foreach *constraint* i *in ILP model* **do**
  if *constraint* i *is modifiable* **then**
    Introduce elastic variable e_i;
    Assign penalty weight p_i from P;
    Modify constraint i to include e_i;
  else
    Keep constraint i unchanged;
  end
end
Set Elastic Objective: Minimize $Z = \sum_i p_i e_i$;
Solve modified ILP model with Elastic Objective;
if *Solution is Feasible* **then**
  Analyze the values of e_i to determine which constraints are violated and by how much;
  Calculate the total cost of violations;
end

TABLE I
SFC RESOURCE DEMANDS

Resource	Units
CPU	1-4 cores
Memory	1-8 GB
Storage	100-200 GB
Processing delay/VNF	60-100 μs
Propagation delay/vlink	60-400 μs
Bandwidth/vlink	10-20 $Mbps$

constraints could be feasibly relaxed and to what degree, demanding detailed analysis and interpretation.

VI. PERFORMANCE EVALUATION

In our experiments, we use the three SFC topologies presented in [18] and shown in Fig. 1. We refer to the SFC topologies as Type 1 (blue color), Type 2 (green color), and Type 3 (red color). The computational demands for the VNFs range between $1-4CPU$ cores, $1-8GB$ of memory, and $100-200GB$ of storage. Each VNF, according to its type, has a processing delay requirement between $60-100$ μs. Finally, each virtual link has propagation delay requirements in the range $60-400$ μs and bandwidth requirements in the range $10-20$ Mbps. The SFC and VNF characteristics are summarized in Table I.

Real topological characteristics have been used from a Ciena infrastructure covering a metropolitan network for the physical network. As shown in Fig. 4, the network consists of 169 nodes. Light blue nodes denote the gateways from which SFC requests can enter the network, blue nodes depict routers, yellow nodes illustrate the edge data centers, and red nodes are the core cloud data centers. We assume that cloud servers are more computationally powerful compared to edge servers. Moreover, we consider a range of physical link bandwidths representative of typical Metropolitan and Wide Area Networks (MAN, WAN). Specifically, the bandwidth of the physical links in our network ranges from 10 $Gbps$ to 100 $Gbps$. This bandwidth range is chosen to reflect the diverse capacities encountered in real-world network infrastructures, allowing us to evaluate the performance of our VNF placement and routing algorithms under conditions indicative of actual operational environments.

In the following, we perform three sets of experiments. In the first set (*Offline Planning Experiments*), we primarily focus on evaluating the performance of our offline planning algorithms (ILP, SeqSort and SeqBiased) regarding execution time, deployment cost, and acceptance ratio. The objective is to establish a baseline for the efficiency and scalability of these algorithms in a controlled, static scenario. The second set (*Offline Planning with Online Correction Experiment*), building on the previous experiment, aims to demonstrate the practical applicability of the algorithms in a dynamic environment. Here, we incorporate the aspect of real-time adjustments to the pre-allocated SFCs, highlighting how online corrective actions can enhance performance metrics such as blocking probability and execution time. Finally, in the third set (*Infeasibility Restoration Planning Experiment*), we assess the performance of the infeasibility restoration techniques regarding repair time and cost. This experiment is designed to address scenarios where the initial planning (offline or online) encounters infeasibilities. The focus is on assessing the effectiveness of our proposed infeasibility restoration techniques (ShuffleFilter, CostBased, and ElasticHeuristic) in adapting to network constraints by suggesting optimal resource additions. The evaluation parameters used in our experiments are summarized in Table II.

A. Offline Planning

In Section IV, we presented the three offline approaches for the infrastructure planning, namely, ILP, SeqSort, and SeqBiased. The ILP method is solved using the Gurobi Optimizer, a highly efficient and powerful solver for linear programming (LP), mixed-integer linear programming (MILP), and other optimization problems [21]. The Gurobi Optimizer provides state-of-the-art algorithms and parallel processing capabilities, making it one of the best choices for solving ILP problems.

To assess the performance of these proposed approaches, we compare against a Grey Wolf Optimizer (GWO) approach, presented in [16]. Similar to our proposed framework, the GWO tries to first find the allocation of a batch of SFCs in a network. Likewise, it also tries to search for an optimal or near-optimal solution that minimizes the overall cost while maximizing the allocation success ratio. Fig. 5 presents the execution time for the four algorithms for an incremental number of SFC requests. As expected, the execution time of the global ILP increases exponentially with the number of SFCs. This behavior corroborates the added complexity of the ILP approach, especially when the experimentation setup grows larger. As a reminder, the ILP method solves the problem for all SFCs, as a batch, at the same time. On the other hand, the SeqSort and SeqBiased algorithms, illustrated by the blue and green curves, respectively, exhibit a linear growth in the execution time. In addition, their execution times are much

TABLE II
EVALUATION PARAMETERS SUMMARY

Parameter	Description
Execution Time	Time taken by an algorithm to allocate SFCs.
Deployment Cost	Objective cost incurred in deploying SFCs
Acceptance Ratio	Proportion of successfully allocated SFCs in online VNF allocation
Blocking Probability	Likelihood that an incoming SFC request will be rejected
End-to-End Delay	Average time for data travel in the network from SFC source to destination
Repair Cost	Cost for adding resources to restore feasibility
Repair Time	Time taken for infeasibility restoration
Allocation Success Ratio	Sequential placement algorithm's efficiency in allocating batch of SFCs
Online Computation Time	Time taken for dynamic computational adjustments when preallocated solutions are exhausted
Offline Execution Time	Static computational time taken for preallocating SFCs before their actual arrival

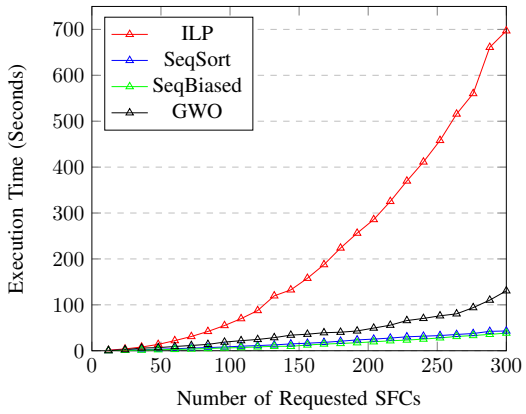

Fig. 5. Average execution time analysis.

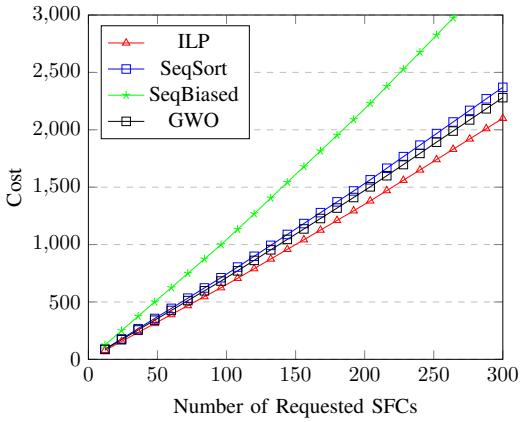

Fig. 6. Cost analysis (Ample available resources).

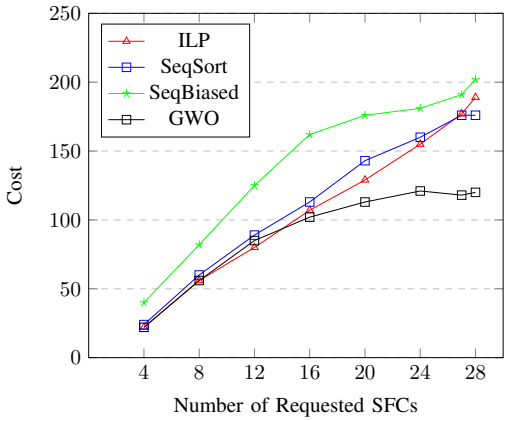

Fig. 7. Cost Analysis (Limited available resources).

lower than ILP, signifying that they are far more scalable and better suited for larger scale problems. This is due to the fact that these algorithms perform the allocation of only one SFC at a time. Another interesting observation is that although the SeqSort is based on a reduced ILP formulation (i.e., only for one SFC at a time), it manages to achieve execution times close to SeqBiased, a purely greedy heuristic.

Like SeqSort and SeqBiased, the black curve illustrating the GWO algorithm displays a relatively progressive increase in execution time as the number of instances increases. This shows that GWO performs well in terms of execution time and can also be considered as a scalable option. Nonetheless, the results are encouraging for all four algorithms. For instance, the global ILP method, even for 300 SFC requests, it will successfully allocate them in approximately $11min$. This is considered fair for a 169-node infrastructure. Additionally, we have to consider that the particular algorithm is not expected to be used as an online one but rather as a offline planning tool for the network operator. Hence, even if an operator re-plans the network on a daily or even hourly basis, the offline algorithms can promptly provide the necessary allocations to be made. The results are even more promising for the two other algorithms providing 300 allocations in less than a minute ($0.2s$ per SFC). Thus, they can be used for larger network infrastructures, where the global ILP is expected to take notably more time. The GWO stands in the middle with an execution time that is worse than SeqSort and SeqBiased but considerably less than the ILP.

Fig. 6 presents the objective cost for the four algorithms for an incremental number of SFC requests, showing the cost comparison for up to 300 requests for a network with ample communication and computational resources capable of handling an excess of 300 requests, which is the same setup used to generate the results in Fig. 5. At a glance, we observe that the ILP yields the lowest cost, followed closely by SeqSort and GWO, while SeqBiased is the most inefficient. Since the network is capable of handling an excess of 300 requests, in this case all methods were able to allocate up to 300 requests successfully in this experiment, i.e., the success ratio is equal to 1 for all of them. Fig. 7 illustrates the objective

function's cost for the four algorithms when the number of SFCs ranges between 4 and 28. To make the experimentation more challenging, this time we limit the network's resources to accommodate at most 28 requests. As the global optimal, we see that the ILP obtains again the lowest cost among the other algorithms, confirming that it delivers efficient solutions. At the same time, it can be observed that SeqSort provides solutions very close to optimal for the different numbers of SFCs considered. This is a propitious behavior as SeqSort also showcases low execution times. In contrast, SeqBiased is proven to be less effective, as it shows a steeper increase in cost as the number of SFCs increases. The GWO algorithm displays a consistent increase in cost as the number of SFCs grows, closely approximating the ILP's cost. Nonetheless, after 12 SFCs, this behavior changes, with GWO demonstrating a lower cost than ILP. To better understand this behavior, we must draw our attention to Fig. 8, where the allocation success ratio is depicted; for up to 12 SFCs, all four algorithms manage to find a proper placement solution. However, after this point, GWO (for 16 SFCs), SeqBiased (for 20 SFCs), and SeqSort (for 24 SFCs) do not manage to accommodate all the requests. In contrast, the ILP succeeds in finding a possible allocation for all SFC combinations, even for the configuration with the 28 SFCs, where we have deliberately dimensioned our network to only have one feasible solution. Thus, the ILP method, maintains a constant allocation success ratio of 1.

SeqSort also maintains an allocation success ratio of 1 for most of the range. However, the success ratio slightly decreases to 0.96, 0.95, and 0.92 for 24, 27, and 28 requested SFCs, respectively, suggesting that the performance deteriorates as the number of requested SFCs increases. Similarly, SeqBiased begins with a success ratio of 1, but as the number of requested SFCs rises, the success ratio gradually declines. This suggests that SeqBiased struggles to allocate all of the requested SFCs when the network is more stressed. Finally, the GWO algorithm maintains a success ratio of 1 when dealing with up to 12 SFCs. After that, however, it suffers from a substantial decline in the success ratio as the number of SFCs increases, indicating that GWO is not as practical as ILP and even SeqSort in successfully allocating all requested SFCs when the problem scales up. Thus, as the acceptance ratio falls, so does the cost function since fewer SFCs are being accommodated, which justifies the behavior noticed in Fig. 7. Fig. 7 - Fig. 9 present the trade-off between placement efficiency and execution time, and how an InP can benefit from both a global and/or a sequential offline approach.

Fig. 9 compares the four methods under consideration in terms of how the average E2E delay changes as the number of requested SFCs increases. For the ILP, the average E2E delay remains relatively low and stable throughout the tested range of requested SFCs. For SeqSort, the average E2E delay increases moderately as the number of requested SFCs grows. On the other hand, for SeqBiased, the delay significantly increases with the growth of requested SFCs compared to ILP and SeqSort methods. Finally, for GWO, the delay increases notably as the number of requested SFCs increases, particularly after having more than 20 requested SFCs. From this figure, it is evident that the ILP method

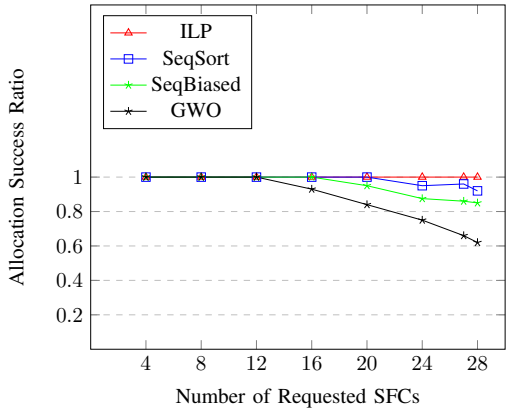

Fig. 8. Success Ratio analysis.

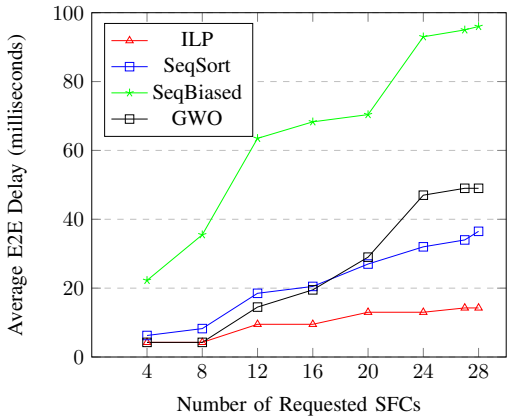

Fig. 9. Average E2E delay analysis.

results in the lowest average E2E delay across all volumes of requested SFCs, followed by SeqSort. On the other hand, the SeqBiased and GWO methods exhibit higher delays, with SeqBiased having the highest average E2E delay among the four methods. Overall, the global offline ILP can provide the best performance in terms of cost, success ratio, and E2E delay, while SeqSort is a viable alternative, approximating well the performance of ILP while keeping the execution times low. This makes the particular algorithm a good and fast network planning tool candidate for an InP.

B. Offline Planning with Online Correction

In this second set of experiments, we use the global ILP and GWO approaches to offline plan the placements (pre-allocations) of the VNFs. The global ILP method guarantees global optimality, but as the input's size increases, the computational time grows exponentially, making it impractical for large-scale scenarios. Therefore, we also consider the GWO approach for offline planning, which converges faster than the ILP for large-scale problems but may provide near-optimal solutions instead. In this case, we assume the operator expects to receive 28 SFCs from specific gateways. Specifically, the 28 preallocated solutions are equally distributed over the four gateway routers w_2, w_3, w_4, and w_5, as shown in Fig. 4. Hence, at each gateway router, we can access seven preal-

TABLE III
DEPARTURE AND ARRIVAL RATES PER SFC TYPE PER GATEWAY ROUTER

	λ	μ
Type1G2	0.23504524621	0.29850746268
Type2G2	0.37500937523	0.29850746268
Type3G2	0.29850746268	0.29850746268
Type1G3	0.23504524621	0.29850746268
Type2G3	0.37500937523	0.29850746268
Type3G3	0.29850746268	0.29850746268
Type1G4	0.23504524621	0.29850746268
Type2G4	0.37500937523	0.29850746268
Type3G4	0.29850746268	0.29850746268
Type1G5	0.23504524621	0.29850746268
Type2G5	0.37500937523	0.29850746268
Type3G5	0.29850746268	0.29850746268

located solutions as follows: 3 of Type 1 (blue color), 2 of Type 2 (red color), and 2 of Type 3 (green color). Later, we proceed with the actual and online incoming SFC requests. For this dynamic allocation, a traffic generation model was used to represent the arrival and departures of each type of SFC at each gateway. Specifically, following the literature, the interarrival time of each type of SFC at each gateway follows a Poisson distribution, while the lifetime of each SFC follows an exponential distribution. The departure and arrival rates used are shown in Table III, while the simulation lasts for 10000 events (arrivals and departures). In this table, λ represents the arrival rate of service requests, and μ the service rate at which these requests can be processed. The inclusion of these parameters is intended to provide insight into the network's capacity and performance under different load conditions.

Based on this modeling, the efficiency of the three approaches presented in Section IV-B is assessed in terms of blocking probability and execution time. The execution time is broken down to offline execution time (static computational time), accounting for the average execution time per SFC during the offline allocation, and online computation time (dynamic computational time), accounting for the average execution time when an incoming SFC cannot fit any available preallocation and a new solution must be found in real-time. In this set of experiments, the available capacity of the physical infrastructure was set to accommodate more than 28 SFCs, in order to show that even when a poor estimation of the total simultaneous SFCs that will coexist is made, the corrective actions of the online placement can remedy the situation. Besides, it is hard to precisely determine the exact number of preallocated solutions we can fit in the network, as it depends on the SFC demands and their distribution.

Fig. 10 illustrates the results, where the left y-axis represents the percentage of the blocking probability and the right y-axes the execution times in seconds. This figure shows the results of using four approaches; the first two methods (ILP Pre-Allocations and GWO Pre-Allocations) are meant for offline preallocation, using the global ILP and GWO, respectively; the other two methods cater for offline planning with online correction. The ILP Pre-Allocations & online-ILP uses the single SFC ILP method for online correction, whereas the ILP Pre-Allocations & Greedy approach uses the greedy method, as presented in Section IV-B. The two preallocation methods showcase the worst blocking probability since when an incoming SFC cannot fit to a preallocated solution, it gets rejected. GWO Pre-Allocations has a higher blocking probability than ILP Pre-Allocations because the former found only 18 pre-allocations instead of 28. It should be noted that if the operator had made an estimation of the arrival of the SFCs with an accuracy of 100%, the blocking probability would be 0% for the ILP Pre-Allocations, as shown in Fig. 8. However, we wanted to show how combining offline and online algorithms could resolve an inaccurate prediction. Regarding the execution time, the online part practically accounts for $0s$ since no algorithm is executed, but rather the solution of the offline allocation is used; the average offline execution time per SFC is around $1.48s$ for ILP, and $0.65s$ for GWO. In contrast, when employing the preallocation plus the online ILP or the Greedy SeqBiased heuristic, the blocking probability is considerably reduced by more than half. In particular, for the ILP preallocation & online ILP mechanism, the blocking probability is reduced by 54% while adding an average execution time overhead of only $0.17s$ per SFC. Similarly, the ILP preallocation & Greedy mechanism reduces the blocking probability by 46% with an average online execution overhead time of $0.12s$. Interestingly, the online ILP version can efficiently find an alternative solution when all pre-allocations are used at practically the same time as the greedy approach.

C. Infeasibility Restoration Planning

In this final part of the evaluation, to assess the performance of infeasibility restoration mechanism, we consider the applicability of the model when a bulk number of SFCs have to be planned before their actual arrival. As stated in Section V, the infeasibility restoration guides the operator on where additional resources are needed to be added in order to accommodate the total anticipated demand. Fig. 11 compares the three proposed algorithms, namely, ShuffleFilter, CostBased, and ElasticHeuristic, for an increasing number of SFCs, while using the network topology called Reduced-50, a reduced version of Metro-169 with 50 nodes [25], as shown in Fig. 4. Here, the x-axis represents the number of SFCs to be planned, ranging from 1 to 50, with a step of 10, and the y-axis represents the repair cost. The SFCs are of different types and are equally distributed across the four gateway routers of the network. To take an extreme scenario, we have set the physical network to be able to accommodate only one type of SFC. Hence, only when 1 SFC is planned to be allocated will the ILP produce a feasible solution. The model will be infeasible for all other combinations, and the three algorithms should suggest where to add the necessary resources. As expected, the ElasticHeuristic provides the best repair cost. What is interesting is that the ShuffleFilter algorithm can also find a repair with a cost very close to the optimal. Additionally, as the number of SFCs increases, the repair cost increases relatively linearly. In contrast, the CostBased heuristic presents the highest cost and the worst performance since it cannot properly assess the impact of a violated constraints in a set of IISes, the way ShuffleFilter does.

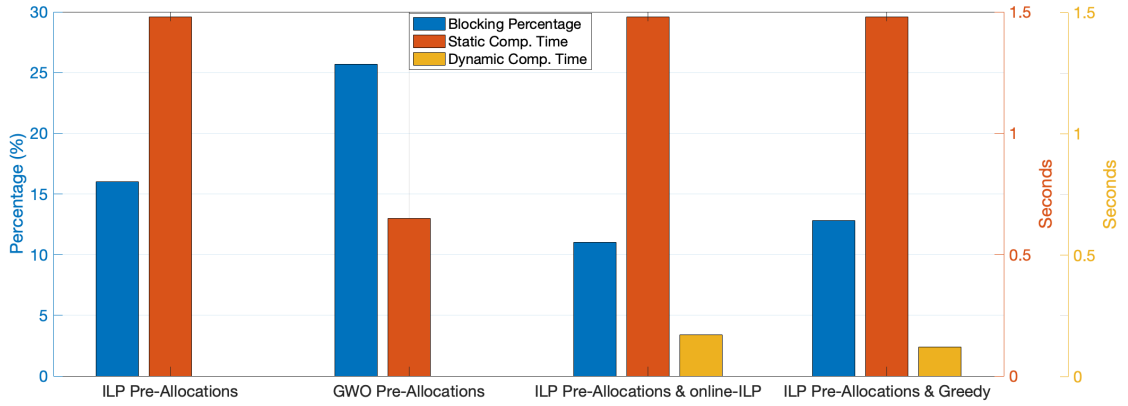

Fig. 10. Simulation results for online assisted pre-allocations.

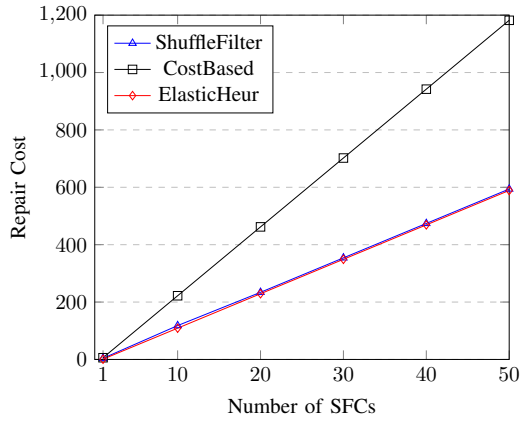

Fig. 11. Repair Cost of ShuffleFilter VS CostBased VS ElasticHeuristic.

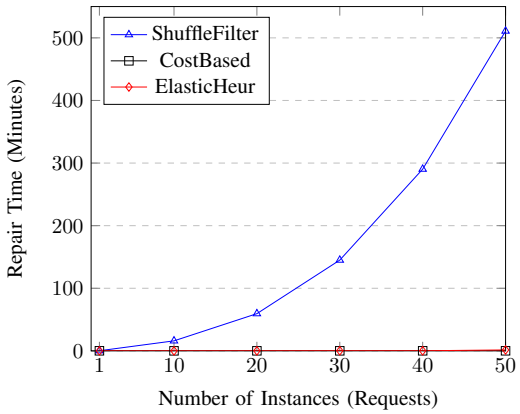

Fig. 12. Repair Time of ShuffleFilter VS CostBased VS ElasticHeuristic.

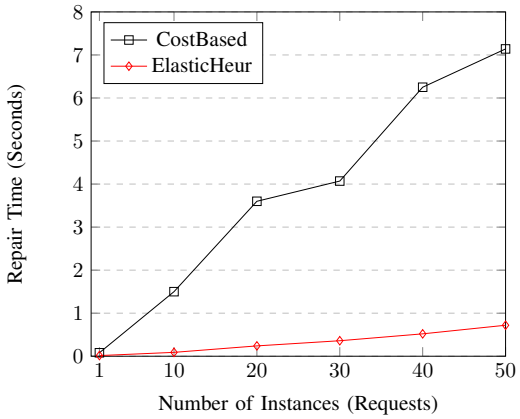

Fig. 13. Repair Time of CostBased VS ElasticHeuristic.

In terms of repair time, as illustrated in Fig. 12, ShuffleFilter proves to be the most computationally expensive algorithm. This was expected since this algorithm must find multiple IISes for each iteration, meaning that the ILP algorithm had to be rerun several times at each cycle, which resulted in a very high execution repair time. Moreover, as the number of instances increases, its repair time increases steeply, indicating that it is less suitable for large-scale problems than the other two algorithms. In contrast, this time, the CostBased approach achieves considerably lower repair times, showing that it is highly scalable. Hence, these two IIS repair approaches create a trade-off between the repair cost and repair time. Once more, the ElasticHeuristic produces the best solution in the least amount of time since the ILP is being run only once and simultaneously indicates where the additional resources should be added. Since in Fig. 12 the high repair times of ShuffleFilter do not allow for a good assessment of the difference between CostBased and ElasticHeuristic, we re-plot the figure only for the two last algorithms, as shown in Fig. 13. As seen here, ElasticHeuristic is much faster and efficient than CostBased, while the execution time gap between them increases with the number of SFCs.

After validating the efficiency of the ElasticHeuristic in terms of both repair time and repair cost, we evaluate how the algorithm behaves for different infrastructure sizes. Specifically, Fig. 14 is a graph comparing the repair time for three different network topologies, Metro-169, Reduced-50, and Abilene-12, a small network with only 12 nodes [25], using the ElasticHeuristic algorithm. As expected, the Metro-169 network, has the highest repair time among the three networks. As the number of instances increases, its repair time increases relatively steeply, indicating that repairing the infeasibilities in this more extensive metropolitan network takes significantly longer than in the other two networks. However, the total

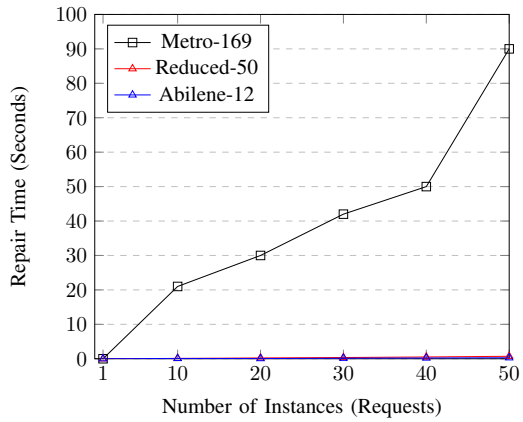

Fig. 14. Repair Time of Metro-169 VS Reduced-50 VS Abilene-12 using ElasticHeuristic.

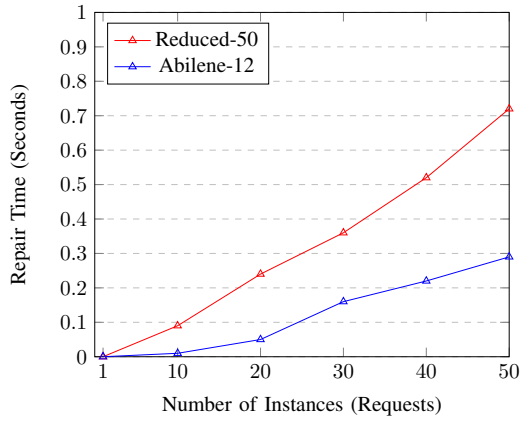

Fig. 15. Repair Time of Reduced-50 VS Abilene-12 using ElastiHeuristic.

time needed is quite short, with the algorithm being able to provide a repair solution in only $90s$. We should not forget that this tool is an offline planning tool that provides feasibility restoration recommendations to the provider. To better grasp the added complexity of restoring the feasibility when moving from a small network, such as the Abilene-12 to a medium-sized network, such as the Reduced-50, we re-plot the figure only for the two last topologies, as shown in Fig. 15. The graph demonstrates that the ElasticHeuristic needs approximately half the time for the Abilene-12 infrastructure than for the Reduced-50 infrastructure.

VII. Conclusion and Future Work

VNF placement in edge/cloud environments has become increasingly popular, leading to the need for efficient service function chaining placement mechanisms. In this paper, we proposed a three-part holistic network planning tool for SPs and InPs that aims to establish versatile solution strategies for VNF placement and feasibility repair that considers offline planning, online corrective actions, and suggestions for locating and repairing potential placement infeasibilities. Specifically, first, an offline, proactive SFC placement mechanism was developed that optimally allocates the physical resources of the infrastructure based on expected workload demands. The mechanism aims to minimize the number of Edge servers utilized for VNF placements, as well as the communication cost, in terms of number of links used to interconnect them. Following, we devised two heuristic algorithms to solve the formulated ILP problem to avoid computationally intractable situations. This offline mechanism was followed by an online, reactive SFC placement mechanism that corrects potential mispredictions in the incoming workload. Finally, a novel infeasibility restoration mechanism was implemented to complement the offline SFC planning step, alleviating resource mismatches between the infrastructure's total capacity and the requested resources of the expected SFC requests. Three approaches were examined for this component, aiming to minimize the induced capital and operational expenses. Extensive simulations demonstrated the effectiveness and efficiency of the proposed mechanisms through numerical results under different configuration scenarios. The feasibility restoration tool was also benchmarked on its accuracy and cost minimization. Future work will focus on complementing the proposed planning tool with machine learning techniques that will learn SFC request patterns and user behaviors in order to predict the number and location of incoming requests more accurately.

References

[1] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The road towards 6g: A comprehensive survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.

[2] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.

[3] R. Mohamed, A. Leivadeas, I. Lambadaris, T. Moris, and P. Djukic, "Fast resource allocation for virtual network functions chain placement," in *2022 International Telecommunications Conference (ITC-Egypt)*. IEEE, 2022, pp. 1–6.

[4] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108177, 2021.

[5] P. K. Thiruvasagam, A. Chakraborty, A. Mathew, and C. S. R. Murthy, "Reliable placement of service function chains and virtual monitoring functions with minimal cost in softwarized 5g networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1491–1507, 2021.

[6] R. Mohammed, I. Lambadaris, A. Leivadeas, J. Chinneck, T. Morris, and P. Djukic, "Automatic feasibility restoration for 5g cloud gaming," in *ICC 2023 - IEEE International Conference on Communications*, 2023.

[7] V. Tran, J. Sun, B. Tang, and D. Pan, "Traffic-optimal virtual network function placement and migration in dynamic cloud data centers," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 919–929.

[8] Q. Zhang, F. Liu, and C. Zeng, "Online adaptive interference-aware vnf deployment and migration for 5g network slice," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2115–2128, 2021.

[9] C. R. De Mendoza, B. Bakhshi, E. Zeydan, and J. Mangues-Bafalluy, "Near optimal vnf placement in edge-enabled 6g networks," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE, 2022, pp. 136–140.

[10] Y. Mao, X. Shang, and Y. Yang, "Joint resource management and flow scheduling for sfc deployment in hybrid edge-and-cloud network," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 170–179.

[11] H.-W. Tseng, T.-T. Yang, and F.-T. Hsu, "An mec-based vnf placement and scheduling scheme for ar application topology," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021, pp. 1–6.

[12] M. Dieye, S. Ahvar, J. Sahoo, E. Ahvar, R. Glitho, H. Elbiaze, and N. Crespi, "Cpvnf: Cost-efficient proactive vnf placement and chaining for value-added services in content delivery networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 774–786, 2018.

[13] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[14] M. Garrich, J.-L. Romero-Gázquez, F.-J. Moreno-Muro, M. Hernández-Bastida, M.-V. B. Delgado, A. Bravalheri, N. Uniyal, A. S. Muqaddas, R. Nejabati, R. Casellas *et al.*, "It and multi-layer online resource allocation and offline planning in metropolitan networks," *Journal of Lightwave Technology*, vol. 38, no. 12, pp. 3190–3199, 2020.

[15] C. You *et al.*, "Efficient load balancing for the vnf deployment with placement constraints," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[16] X. Wang, H. Xing, F. Song, S. Luo, and P. Dai, "Dynamic multicast-oriented virtual network function placement with sfc request prediction," in *2022 14th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2022, pp. 81–88.

[17] S. Pandey, T. Van Nguyen, J.-H. Yoo, and J. W.-K. Hong, "Edgedqn: Multiple sfc placement in edge computing environment," in *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021, pp. 301–309.

[18] J. Halpern, C. Pignataro *et al.*, "Service function chaining (sfc) architecture," in *RFC 7665*, 2015.

[19] A. Leivadeas, M. Falkner, I. Lambadaris, and G. Kesidis, "Optimal virtualized network function allocation for an sdn enabled cloud," *Computer Standards & Interfaces*, vol. 54, pp. 266–278, 2017.

[20] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet, "Placement and chaining for run-time IoT service deployment in edge-cloud," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 459–472, 2019.

[21] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2021.

[22] J. W. Chinneck, *Feasibility and Infeasibility in Optimization:: Algorithms and Computational Methods*. Springer Science & Business Media, 2007, vol. 118.

[23] ——, "An effective polynomial-time heuristic for the minimum-cardinality iis set-covering problem," *Annals of Mathematics and Artificial Intelligence*, vol. 17, no. 1, pp. 127–144, 1996.

[24] N. Chakravarti, "Some results concerning post-infeasibility analysis," *European Journal of Operational Research*, vol. 73, no. 1, pp. 139–143, 1994.

[25] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0—survivable network design library," *Networks: An International Journal*, vol. 55, no. 3, pp. 276–286, 2010.